Parallel Processing in a Computationally-Intensive Workload

Robert J. Bergeron[1]

Report RND-91-007, October 1991

RND Branch
NAS Systems Division
NASA Ames Research Center
Mail Stop 258-6
Moffett Field, CA 94035-1000

[1] Computer Sciences Corporation, NASA Contract NAS 2-12961, Moffett Field, CA 94035

Parallel Processing in a Computationally-Intensive Workload

Robert J. Bergeron
Computer Sciences Corporation
NASA Ames Research Center
Moffett Field, CA 94035, USA

Abstract

This paper presents Cray Y-MP performance results for a computationally intensive workload employing autotasking. The results indicate the 6.0 release of the Cray UNICOS operating system improves the efficiency of autotasked programs. This release reduced parallel processing overhead by reducing the time spent waiting by slave processes for parallel work. System throughput increased significantly and the scheduling parameters strongly influenced the magnitude of the increase.

1.0 Introduction

Recent emphasis on parallel processing to achieve improved MIMD (Multiple Instruction Multiple Data) supercomputer performance has required adjustments in operating systems to allow parallel processes to share data. Efficient performance of workloads containing parallel jobs seems to require more modification to ensure that parallelism does indeed deliver its promised performance gains. Performance improvements in workload environments occur when the operating system efficiently distributes idle CPUs to processes requesting multiple CPUs.

Version 6.0 of the Cray UNICOS operating system claims a significant throughput improvement for workloads executing multiple CPU programs. Factors contributing to the improvement include processor synchronization performed by a software semaphore and processor scheduling performed by an enlightened kernel. This paper reports on performance improvements of UNICOS 6.0 relative to UNICOS 5.1 on heterogeneous workloads characteristic of the Numerical Aerodynamic Simulation (NAS) Y-MP. The remainder of this section describes sources of workload idle time, methods of exploiting such idle, and the factors affecting throughput of parallel jobs in versions 5.1 and 6.0 of the UNICOS operating system.

1.1 Sources of Workload Idle Time

During the course of executing its component jobs, any workload executing in a multiprogramming environment on a multiple-CPU machine will experience a period during which its CPUs will be unable to perform useful work. Such idle time arises for several reasons:

- a user’s I/O operation idles a CPU until the data transfer completes;
- swapping of user jobs by the operating system idles the CPU;
- there are insufficient jobs in the system to keep the CPUs busy;
- the jobs attempting execution oversubscribe memory;
- a critical I/O resource such as the SSD is reserved.

1.2 Methods for Reducing Workload Idle Time

The tuning of specific operating system parameters can control idle time by scheduling the most critical resources and by directing users away from scarce resources. The treatment of any residual idle time is an administrative issue. To transform idle cycles into useful work, one site may choose low priority jobs as “cycle suckers”, whereas another site may choose parallel processing.

Low priority jobs will attempt to begin execution whenever the idle exceeds some limiting condition. These jobs may be unable to execute if oversubscribed memory is the cause of the high idle. Implementation of a queue to meter the low priority jobs by memory may increase the potential for idle reduction.

Cray's implementation of parallel processing exploits periods of residual idle by permitting idle CPUs to assist in the execution of FORTRAN DO loops. Such loop-level parallel processing requires compiler directives. Autotasking is the automatic insertion of these directives by a preprocessor; microtasking is the insertion of equivalent loop-level directives by the user (Cray, 1988). Autotasking and/or microtasking can improve workload throughput by converting idle CPU cycles into cycles performing useful work. The parallel job consists of a master process which executes throughout the job and slave processes which execute according to the demands of the master and constraints of the operating system.

The parallel job demands special attention from the operating system in the areas of process synchronization and process deadlock (Reinhardt, 1985). Synchronization of the CPUs, i.e., the proper scheduling of the arithmetic operations performed by the master and slave tasks, typically involves logical conditions or locks to prevent additional calculation until a condition is fulfilled. The Cray architecture provides shared registers for inter-CPU transmittal of the status of arithmetic operations. The operating system must support the inactive slave tasks, either by allowing them to sleep or by keeping them connected to their CPUs.

Process deadlock occurs when a process is waiting for a condition that will never occur. For example, slave processes executing DO loop iterations always wait at the synchronization point, i. e., the bottom of a DO loop, until the master task signals that the DO loop iterations are complete. If the operating system preempts the CPU executing the master task, the slave processes must wait until the master task unlocks the lock. If the preempted CPU remains busy with other work, the operating system will disconnect all slave CPUs and the calculation will have to be restarted from the point at which the master was interrupted. The operating system must provide deadlock detection and interruption. A workload environment with many high priority tasks interrupting the parallel jobs may display large numbers of deadlock interrupts.

Versions 5.0 and 5.1 of UNICOS displayed large overheads and unpredictable speedups for autotasked jobs executing in a workload (Carter, 1990). These drawbacks encouraged NAS to choose low priority jobs to reduce residual idle.

1.3 UNICOS 5.1 Autotasking

Problems associated with 5.1 autotasking in a workload environment involve both the synchronization and the scheduling of the slave processes onto CPUs.

UNICOS 5.1 employs a hardware semaphore as the test condition for synchronizing the CPUs. The 5.1 implementation provides fast access to shared resources, allowing efficient processing even for small-granularity tasks. However, this approach keeps the user-requested processors executing the semaphore wait test during periods of singletasked activity. A workload environment containing many processes will display a performance degradation when CPUs are performing semaphore tests instead of useful work.

A limited information flow between the 5.1 kernel and the autotasked processes belonging to a single job prevented efficient scheduling. Deadlock delays occurred when the operating system interrupted one of the CPUs executing an autotasked process. These delays arose because the UNICOS kernel forced sibling autotasked processes to wait until the CPU interrupted by the kernel became available for completing its suspended task. Such inefficient use of the CPUs contributed to poor performance in workloads.

1.4 UNICOS 6.0 Autotasking

UNICOS 6.0 employs a software semaphore to limit the amount of time an inactive CPU remains attached to an autotasked job. When the inactive period exceeds a specified value, the autotasked process on the inactive CPU reschedules itself back to the operating system, thus freeing the CPU for useful work. This approach reduces the overhead (CPU cycles) associated with synchronization since multiple CPUs do not remain connected while the master executes singletasked code.

A two-way information flow between the 6.0 kernel and the autotasked job improves process scheduling by reducing process deadlock. The kernel allows a slave task which was executing on one CPU and interrupted to be completed by another CPU. This procedure reduces the long periods of waiting for a particular CPU to become free to complete its DO loop iterations.

Neither the 5.1 nor 6.0 version of the UNICOS kernel use special information about parallel jobs to schedule the processes. When the master task signals for parallel work, the kernel will place that process at the end of the ready-to-run queue. The placement of slave processes at the bottom of the ready-to-run queue represents one type of scheduling parallel processes in a multiprogramming environment; it may also be possible to place sibling processes closer to the top of the ready-to-run queue and thus further promote elapsed time reductions of autotasked jobs in a workload.

1.5 An Autotasking Experiment

To test the improved performance of the 6.0 system, a synthetic workload consisting of NAS Y-MP production codes has been run in both the 5.1 and 6.0 environments under a variety of conditions. This workload executed on a dedicated machine and the jobs in this workload were the only jobs in the system. The experiment employed the same machine configuration used in NAS Y-MP production time. The following sections describe the modelling of the NAS Y-MP workload, the results of model workload executions on the 5.1 and 6.0 versions of the UNICOS operating systems, a discussion of these results, and some conclusions.

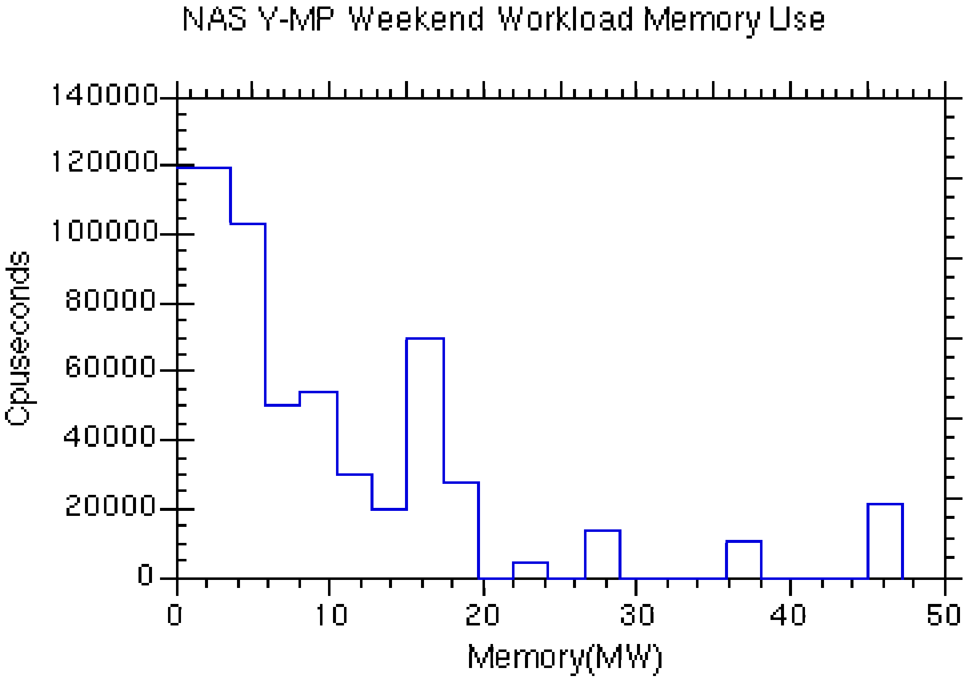
2.0 Modelling NAS Y-MP Workload

The workload, that is, the set of programs submitted for execution, strongly affects the vector performance of supercomputers. The workload will influence the parallel performance of supercomputers even more strongly because parallel versions of typical application codes display a wider range of performance than vector versions. The workload performance of supercomputers executing a set of mixed (single and parallel) programs will also depend upon workload idle time. Such idle reflects administratively set operating system parameters. This section presents the salient features of the NAS Y-MP/8128 workload as it existed in 1990.

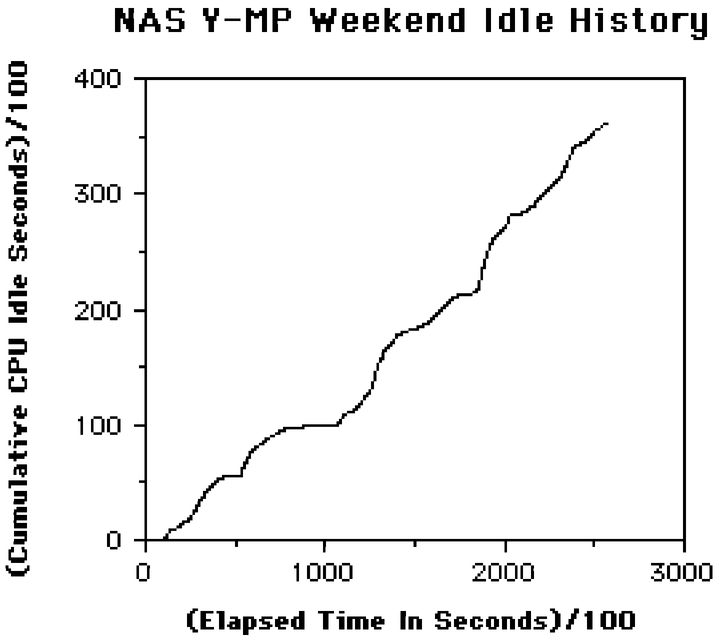
Hardware Performance Monitor (HPM) measurements of the NAS workload performance indicates performance rates of about 90 MFLOPS (Million Floating Point Operations Per Second) per processor (Bergeron, 1990).

Significant idle can occur in off-prime time, defined as evenings and weekends when batch queue submittals dominate execution time. Off-prime batch parameters impose a limit of 12 jobs in execution with a maximum single job size of 64 Megawords (MW). NAS operating system parameters encourage any job equal to or exceeding 16 MW to remain in core until completion. Deferred (low-charge-rate) queues reduce off-prime idle by executing during off-prime periods of high idle.

Examination of the accounting records for a typical weekend provided the following distribution of CPU time as a function of memory size.

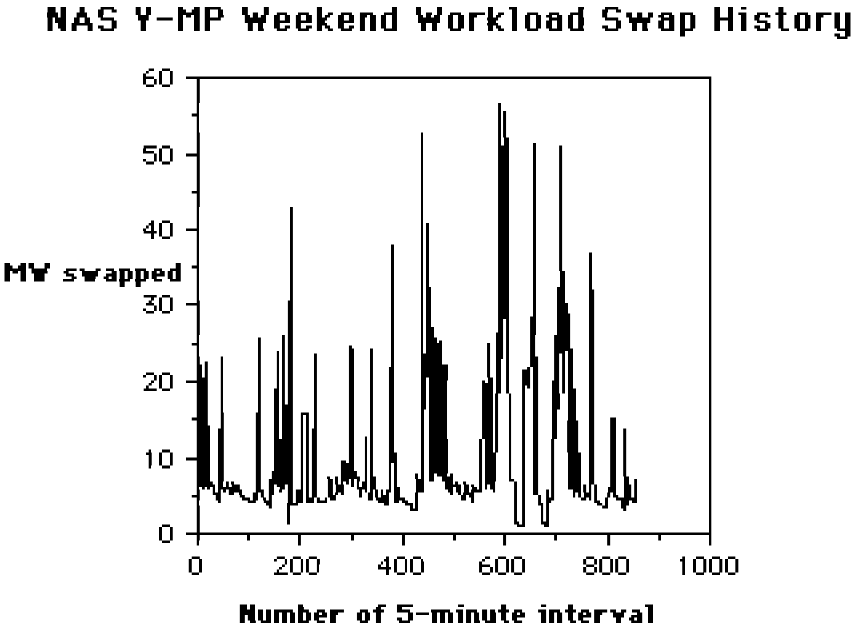


The Cray System Activity Report provided an idle history for the same weekend as shown below.



The slope of the figure is about 1 idle second per 8 elapsed seconds for the weekend shown.

A local utility provided the size of executables swapped to disk for the typical weekend as is shown in the following figure.



A typical weekend has an average of 10 MW swapped to disk and since the first 32 MW of the SSD serves as cache for the swap disk, the swapping I/O takes place at SSD speeds. The oscillations shown in the figure are due to UNICOS time-slicing round-robin scheduling.

The accounting records indicate typical NAS weekend workloads perform with little I/O and little I/O lock time. These records also show that the workload is about 99% singletasked and that the average parallel job is about 10 MW.

The previous remarks indicate that a suite modelling the NAS workload should :

- display a system flop rate averaging 90 MFLOPS,
- show about 12% idle throughout the workload execution,
- display an average 10 MW swapped throughout workload execution,
- consist of 12 jobs executing in batch mode, and
- contain singletasked jobs.

The second and third criteria are related because swapping induced by memory oversubscription can produce CPU idle. Maintenance of a constant swap burden would require a workload involving constant resubmittal of jobs. This approach would, in turn, require a more involved measurement of autotasking performance than elapsed time reduction. Since memory oversubscription produces idle for usage by autotasking, the stronger modelling criterion for these 12 job workloads requires the idle growth rate in the synthetic workload to match that of the NAS Y-MP workload, 1 idle second per 8 elapsed seconds. This latter requirement was adopted and no attempt was made to ensure that the model workloads averaged 10 MW of swap.

To represent autotasking favorably, some of these codes must perform reasonably well on multiple CPUs. Table 1 describes a suite constructed to be consistent with these constraints.

Table 1. Model Workload

Code	CPU (Sec)	Size (MW)	Rate (MFLOPS)	FLOPS %	Comments
C01	595	59	222	11.5	Autotasked
C02	625	55	164	8.9	Singletasked
C03	602	10	161	8.4	Singletasked
C04	601	8	187	9.8	Autotasked
C05	601	8	187	9.8	Autotasked
C06	601	8	187	9.8	Autotasked
C07	617	5	165	8.8	Singletasked
C08	603	7	133	7.0	Singletasked
C09	143	1	122	6.1	Singletasked, 4 copies serially
C10	122	2	125	8.0	Singletasked, 5 copies serially
C11	82	1	131	6.6	Autotasked, 7 copies serially
C12	605	2	105	5.3	Singletasked, SSD sync I/O

The column labelled “FLOPS” denotes the percent of workload floating point operations provided by each code. Consistent with Figure 1, small memory codes make the dominant contribution to the total workload FLOPS.

All codes provide solutions to partial differential equations with the majority involving CFD problems; all codes except C01 and C12 are NAS production codes. Selection of code input parameters attempted to provide about 600 CPU seconds for each code. The table shows that codes C09, C10, and C11 achieve about 600 CPU seconds by resubmittal; in practice, entry of jobs into the batch stream would be a common occurrence.

The total number of floating point operations for this workload is about 1101 billion. The following procedure estimates workload performance:

- Let the first group consist of 8 jobs, one for each CPU, and one of which is C02.
- Let the second group consist of the remaining 4 jobs, one of which is C07.
- Assume that the second group does not start until the first group finishes.
- Assume that the elapsed time for the first group is the CPU time of the longest job, i.e., 625 seconds.
- Assume that the elapsed time for the second group is the CPU time of the

longest job, i.e., 617 seconds.

This procedure gives an elapsed time estimate of 1242 seconds and a system performance rate of 89 MFLOPS per CPU. This estimate neglects system time and efficient scheduling of jobs by the operating system, but it does indicate that the model workload has the potential for agreement with the desired NAS Y-MP rate.

Although the above workload memory requirement is 166 MW, the swapping burden will decrease linearly as the jobs complete and prevent satisfaction of the constant 10 MW swap criterion.

Table 2 shows dedicated 8-CPU speedups and efficiencies demonstrated by the codes designated as autotasked in Table 1 (C03 is included in Table 2 because an autotasked version of this code is used in Section 3.1.2.).

Table 2. Performance of Autotasked Codes in Dedicated Time

UNICOS	6.0		5.1	
CODE	Speedup	Efficiency	Speedup	Efficiency
C01	7.67	.959	7.74	.968
C03	3.75	.469	4.49	.561
C04	3.49	.436	5.54	.693
C05	3.49	.436	5.54	.693
C06	3.49	.436	5.54	.693
C11	3.12	.390	3.90	.488

The autotasking preprocessor, in modifying the FORTRAN source for codes C03, C04, C05, and C06, required special directives to implement in-lining of certain subroutines. The major emphasis in this report was on understanding the workload performance, and additional autotasking effort would have certainly improved the speedups for the autotasked codes. C01 is an exception to this claim since this code already displays a very good speedup. The workload initially contained an autotasked version of C03, but revisions to the autotasker created execution problems for this code. Most of the results quoted in the next chapter include only five autotasked jobs; Section 3.1.2 provides results for a workload with an autotasked version of C03.

A comparison of dedicated performance, as shown in the table, indicates speedup and efficiency penalties for UNICOS 6.0. These reductions reflect the voluntary rescheduling of the CPUs back to the operating system. This feature improves the performance of workloads containing autotasked jobs, but degrades the performance of autotasked jobs in dedicated time. UNICOS 6.0 provides a path for obtaining the 5.1 parallel efficiency with the hardware semaphore, but Cray does not recommend this path for workloads since it degrades workload performance.

Achievement of workload performance gains in dedicated time slots required some revision of the modelling details. Initial UNICOS 5.1 executions of a slightly different workload with no memory oversubscription displayed no autotasking gains. This experience seemed to indicate that autotasking gains required the additional idle generated by the memory oversubscription shown in Table 1. Increasing memory requirements to produce additional idle prevented satisfaction of the 12% idle criterion.

Moreover, initial experience with the NAS scheduling parameters in dedicated time indicated limited swapping for the oversubscribed workload. The scheduler would force execution of one of the two large jobs while keeping the other large job on the swap disk until sufficient memory allowed a swap-in. The NAS scheduling parameters seemed to limit the swapping-induced idle and consequently, autotasking performance gains. Production of idle time in the model workload thus required the creation of a set of scheduling parameters to distribute time slices to large memory jobs on the same basis as the smaller jobs. These "high swap" parameters promoted swapping on a timely basis and created the idle thought necessary for autotasking performance gains. Appendix A provides a comparison of the scheduling parameters used in this analysis.

During execution, background UNIX scripts monitored system idle, workload CPU, memory on the swap disk, and overall system activity.

3.0 Results

This section presents three examples demonstrating the reduction in workload elapsed time under autotasking. The first example (Section 3.1) involves a comparison between UNICOS 6.0 and 5.1 autotasked performance. The second example (Section 3.2) shows the effect of 6.0 scheduler parameters on autotasking performance. Appendix A discusses the UNICOS operating system parameters in more detail. A third example (Section 3.3) provides the performance improvement obtained by maintaining a single job in autotasked mode to utilize idle CPU cycles. The final section presents some discussion of the role of the operating system in supporting autotasking.

3.1 A Comparison of UNICOS 5.1 and 6.0 Performance

These workloads executed in a dedicated environment to provide a controlled investigation of autotasking-generated throughput improvements. Since the workload consisted of a fixed number of jobs, a throughput improvement would correspond to a reduction in elapsed time.

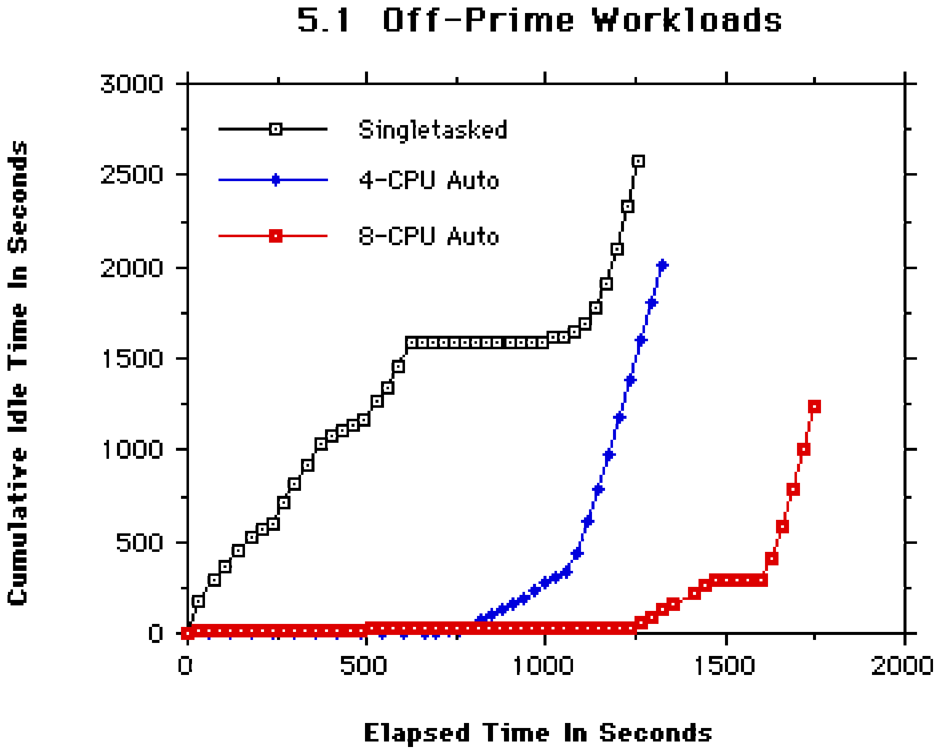
Execution began by invoking a UNIX script which recorded the date, initiated 12 background scripts, paused until all 12 background scripts had completed, and then recorded the completion date. Each of the 12 background scripts initiated execution of a workload job and requested accounting information for the job. Accounting logs provide a wealth of job execution data including elapsed time, CPU time, system time and idle (semaphore wait) time. Executables in the singletasked workload corresponded to singletasked versions of the programs described in Section 2.1. The autotasked workload substituted autotasked executables corresponding to the 5 autotasked programs.

A monitoring script executed every 30 seconds to record a history of the CPU time accumulated by the programs in the workload. The script also provided a snapshot of main memory, swapping information, and system counters.

3.1.1 UNICOS 5.1 Off-Prime Workloads

This section discusses the performance of 5.1 workloads executing with operating system parameters used to administer the NAS off-prime workloads. These off-prime parameters generate a low-swapping, low-idle environment by encouraging the completion of all jobs exceeding 16 MW and by promoting the execution of CPU-intensive jobs. Under the 5.1 system, NAS employed 4 CPUs per process as the default value. The experiments include 5.1 measurements for 4-CPU and 8-CPU workloads to allow comparison with the 6.0 system, which employed an 8-CPU default. The following figure shows the cumulative idle growth for the three cases considered:

- a workload with all jobs singletasked,
- a workload with a maximum of 4 CPUs per job, and
- a workload with a maximum of 8 CPUs per job.



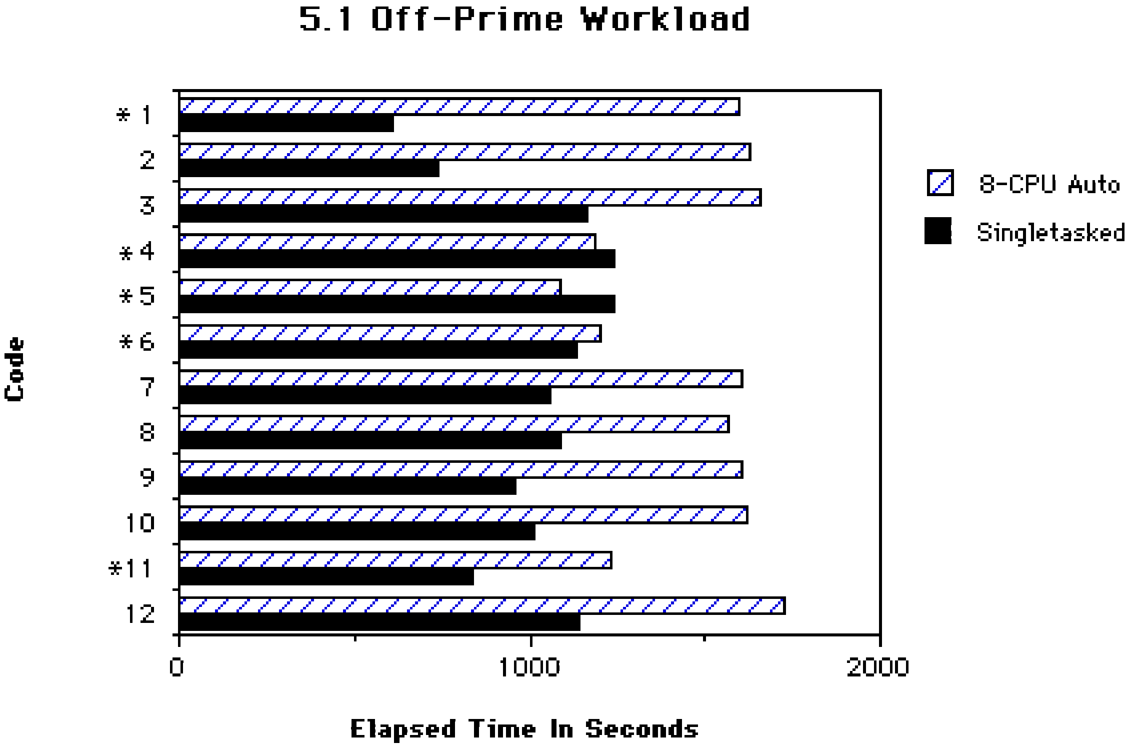
The singletasked workload idle grew at about 2.5 CPU seconds per elapsed second for the first 625 seconds. Idle grew due to CPU memory starvation since the two large memory programs, C01 and C02, occupied the major portion of the memory. Completion of the big jobs began a period of zero idle growth because all jobs then fit into memory. At about 1000 seconds, completion of several more jobs reduced the number of jobs executing to less than 8 and the idle increased at a rapid rate until the workload completed.

The memory starvation due to two large jobs in core is atypical since NAS normally maintains several small jobs in core to alleviate memory starvation. However, the workload did present a good opportunity for autotasking to reduce idle and enhance throughput.

The 4-CPU autotasked workload displayed zero idle for the first 750 seconds of elapsed time and the 8-CPU workload displays zero idle for the first 1200 seconds of elapsed time. However, zero idle does not mean the CPUs perform useful work under the 5.1 system. The CPUs are merely executing the semaphore wait test during much of this time. The accounting logs indicate that about 25% of the elapsed time for the autotasked jobs is spent in the semaphore wait.

Note that the elapsed time for each of the two autotasked workloads exceeded that of the singletasked workload.

The following figure compares the elapsed time of individual jobs in the 5.1 system. The figure indicates the elapsed time for each code for execution in the singletasked and autotasked workloads. Asterisks label the autotasked codes. The figure shows that autotasking reduced elapsed time for only two of the four autotasked codes, C04 and C05, and that all other codes experienced elapsed time increases.

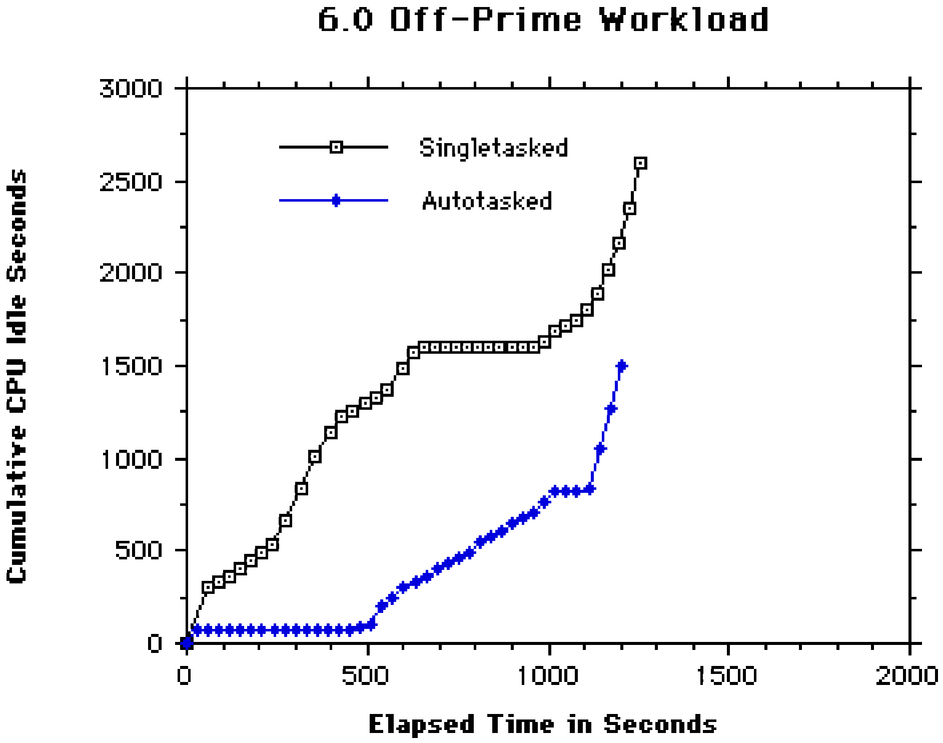


The accounting logs show that all autotasked codes required a much larger amount of CPU time than their singletasked counterparts. For example, C04 required 602 CPU seconds in the singletasked workload and 1720 CPU seconds in the autotasked workload. About 360 seconds of this extra CPU time reflected time spent executing the semaphore wait test. The extra CPU time required for the autotasked jobs prevented the singletasked jobs from performing any work until the autotasked jobs completed.

3.1.2 UNICOS 6.0 Off-Prime Workloads

The following discussion presents the results for the same workload as discussed in Section 3.1.1; however, this time the workload will be executed under the UNICOS 6.0 operating system. Execution of this example occurred several weeks after NAS administrators had obtained a stable set of 6.0 scheduler parameters. Execution of all UNICOS 6.0 workloads use the 8-CPU default for autotasked jobs.

The following figure shows the growth of cumulative idle for the 6.0 Off-Prime workloads.

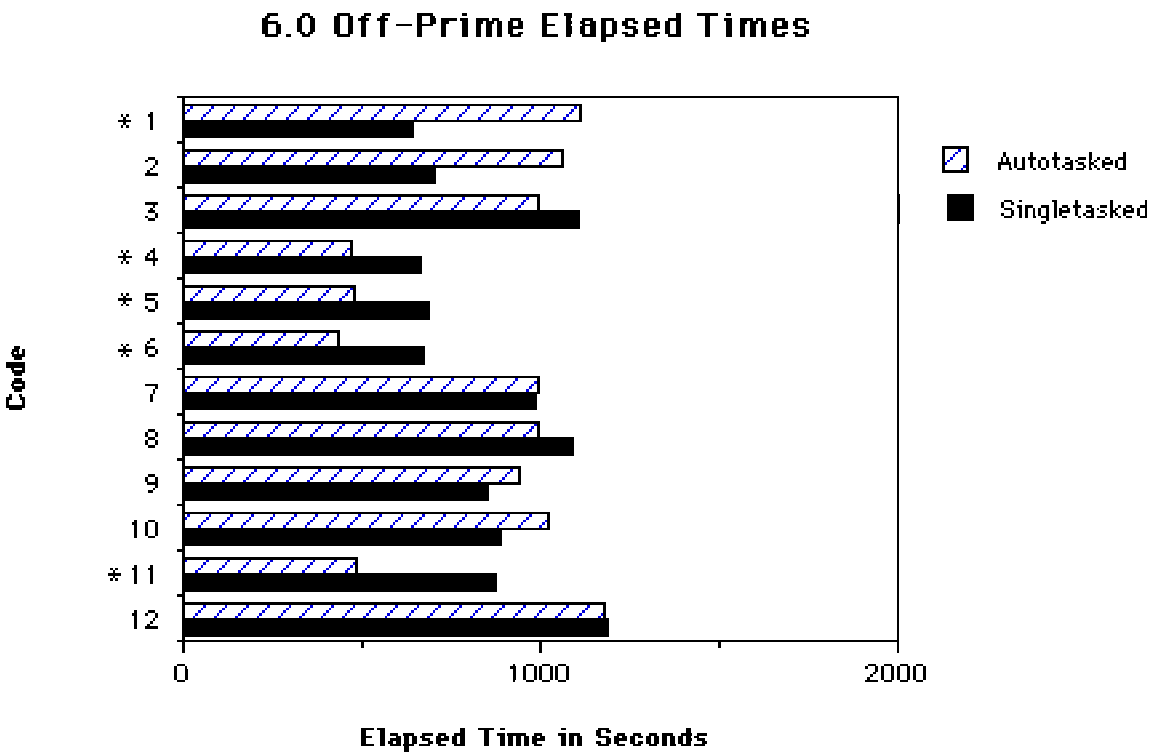


UNICOS 6.0 singletasked idle growth displayed the same rate of increase as the 5.1 case, about 2.5 idle seconds per elapsed second. Completion of one large memory job at 650 seconds began a period of zero idle which continued until the number of jobs executing decreased below the number of CPUs at 1000 seconds. Idle growth resumed as additional jobs completed and freed the CPUs.

The accounting logs indicated that four of the autotasked jobs had completed at 500 seconds, thus verifying that autotasking was responsible for the low idle growth during that time. The idle then grew because there were only seven jobs that could fit into the available memory. Since the job residing on the swap disk was the large memory autotasked job, C01, this situation illustrated how an improvement in the scheduler's awareness would have reduced idle. At about 1000 seconds, idle growth paused as the final large memory autotasked job began execution. Upon completion of the autotasked job, the idle rose again until the final job completed.

The autotasked workload elapsed time was 1173 seconds, 4% less than that of the singletasked workload.

The following figure compares the elapsed times for the 6.0 workloads.The figure indicates the elapsed time for each code for execution in the singletasked and autotasked workloads. Asterisks label the autotasked codes.



The figure shows that autotasking reduced elapsed time for all autotasked codes except C01. The workload oversubscribed memory and the UNICOS scheduler elected to place C01 on swap until completion of other jobs freed sufficient memory for C01.

As was true for the 5.1 case, autotasking alters the order of job completion. Off-Prime parameters disable the UNICOS hog constraints so that once the scheduler connects a process to a CPU, only a kernel or I/O interrupt can disconnect the process. The high efficiency displayed by the autotasked jobs indicated that the number of self-driven I/O interrupts are small; thus these jobs finished first.

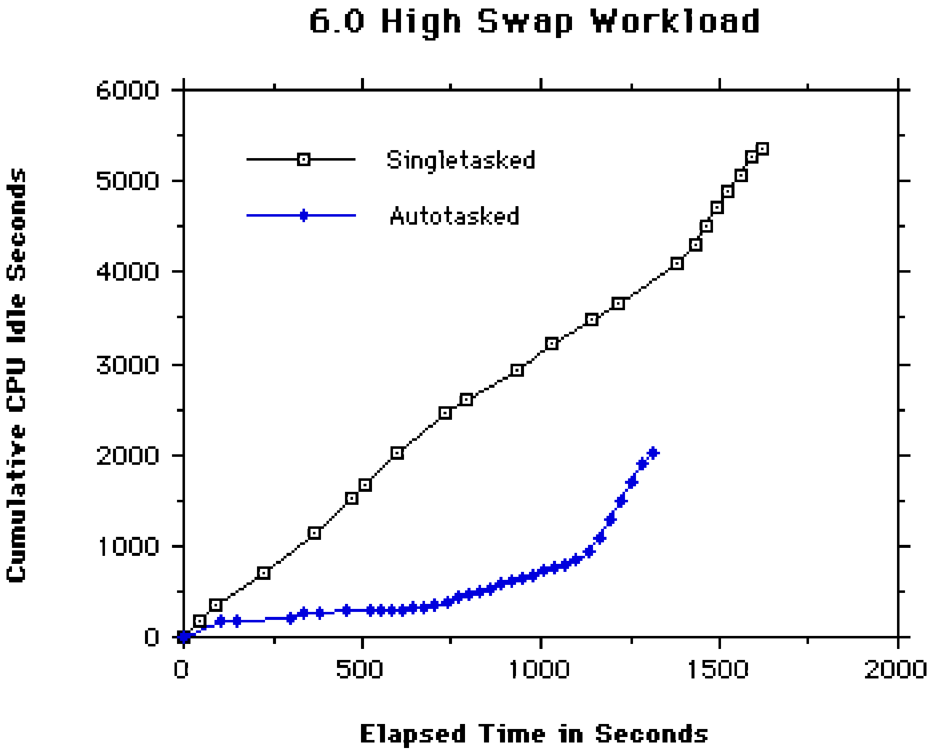
Replacement of the singletasked C03 with its autotasked counterpart increased the number of autotasked jobs to 6. In this workload, 5 autotasked jobs finished before any of the singletasked jobs with the scheduler placing C01 on swap as before. This workload required 1230 seconds. an increase of 5% over the 1173 seconds recorded for the previous workload. The elapsed time increased because one processor was idle during much of the execution of the remaining 7 jobs.

3.2 UNICOS 6.0 Autotasking and Scheduler Effects

This section shows elapsed time and idle for 6.0 cases involving High Swap and Prime NAS scheduler parameters. UNICOS allows modification of its scheduling parameters via an interactive command. Time constraints prevented execution of these cases under the 5.1 version of UNICOS.

3.2.1 UNICOS 6.0 High Swap Scheduling Parameters

The High Swap parameters assign an equal priority to all jobs in the workload. Since the workload oversubscribed memory, this assignment promoted the swapping of executables to disk. Examination of system logs indicated that the scheduler gave all jobs an approximately equal timeslice.

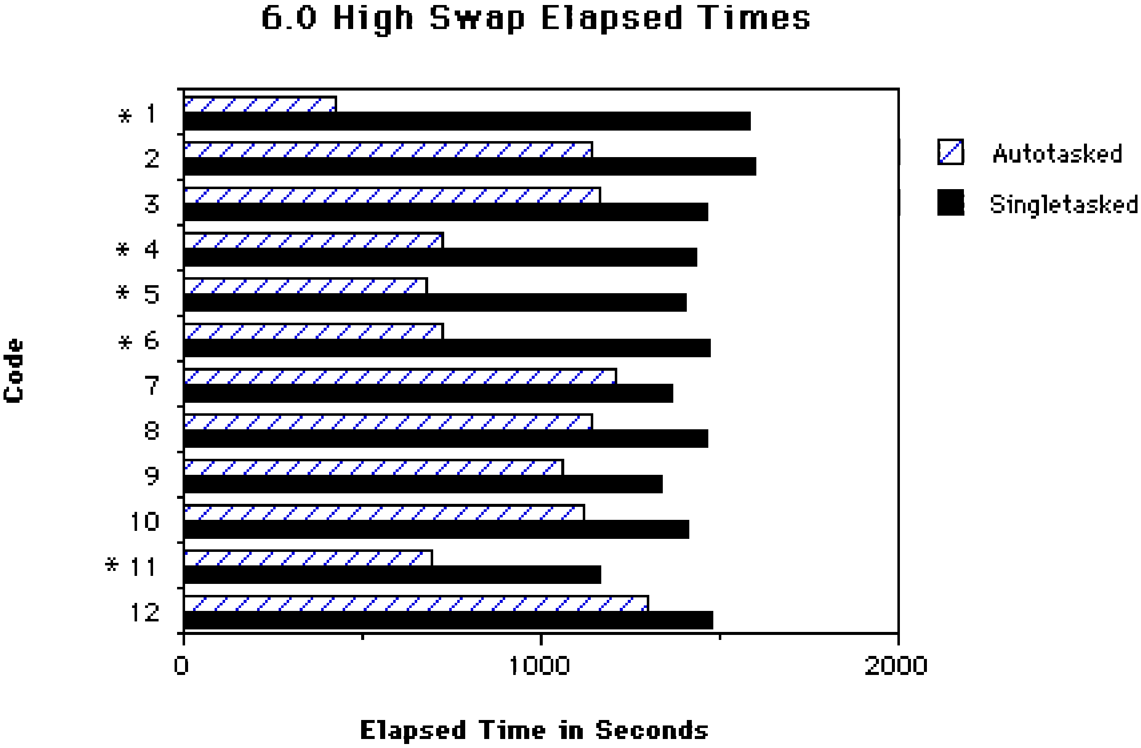


The above figure shows the growth of cumulative idle for the High Swap workloads. For the singletasked workload, idle time grows at a rate of about 3.25 idle seconds per elapsed second until about 1400 seconds. At this time, there are only 8 jobs executing in the system and idle grows much faster as completion of each of the remaining jobs idles a CPU.

For the autotasked workload, idle time grows linearly for about 100 seconds and then grows at a much slower rate until about 750 seconds. Examination of the system logs for the first 100 seconds indicated that the system had created all of the autotasked slave processes; after this time autotasking was able to use idle seconds created by the swapping. Completion of all 5 autotasked jobs at 725 seconds initiated a slow and then rapid increase in idle as the CPUs became free.

Autotasking reduced the elapsed time from 1600 seconds to 1296 seconds, a reduction of 19%.

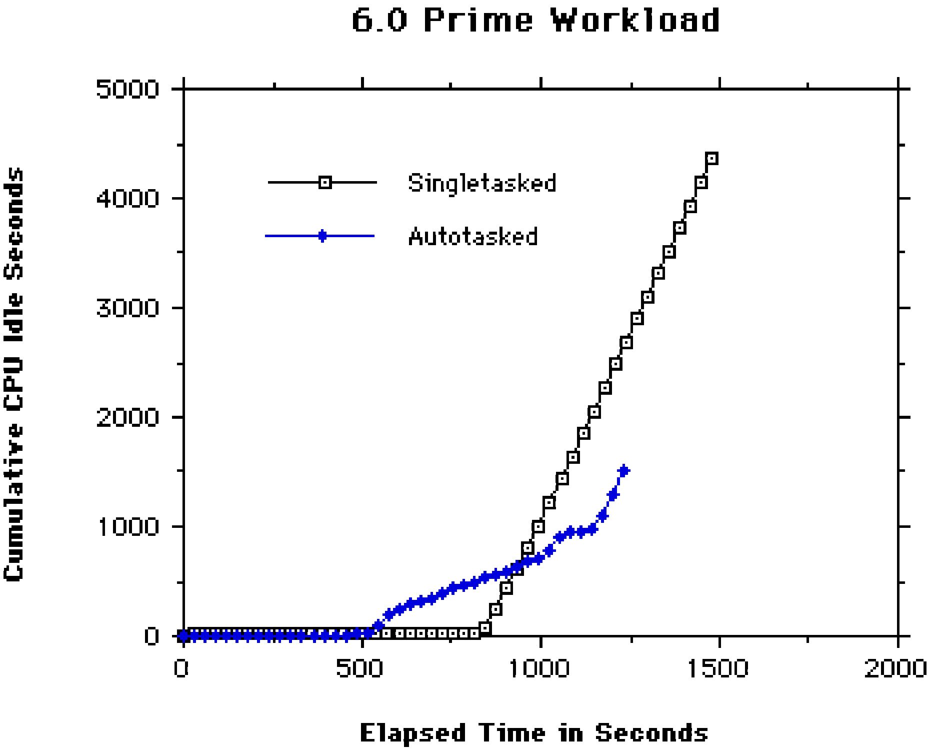
The following figure compares the elapsed time of individual jobs in the 5.1 system. The figure indicates the elapsed time for each code for execution in the singletasked and autotasked workloads. Asterisks denote the autotasked jobs.



The figure shows that autotasking reduced the elapsed time for all jobs, single and autotasked, in the High Swap workload. This result is an example of autotasking providing a clear improvement for workloads with sustained swapping.

3.2.2 UNICOS 6.0 Prime Scheduling Parameters

The second case displaying scheduler effects involves execution of the 12-job workload under NAS prime-time scheduler parameters. These parameters provide a low-idle environment while supporting a strong interactive component. The parameters enforce restrictions on large memory and large CPU jobs. This case addresses the ability of a scheduler devoted to rapid turnaround of small jobs to provide effective autotasking performance.

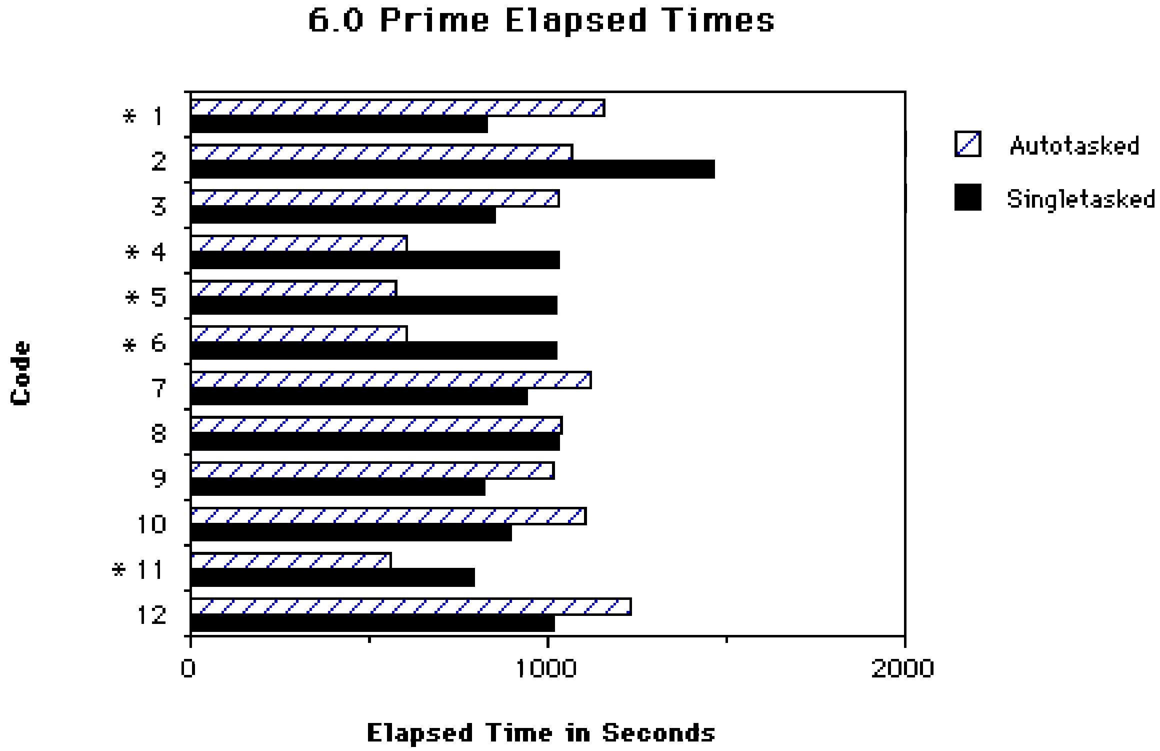


The above figure shows an 850 second period of zero idle for the singletasked workload. The scheduler allowed 11 of the 12 codes to fit in the memory while holding back the 55 MW job, C02. The completion of these jobs in 850 seconds led to an increase in the idle due to an insufficient number of jobs to keep the CPUs busy.

The autotasked workload saw the four smaller autotasked jobs dominate the first 500 seconds of execution. The remaining 8 jobs, including the large memory autotasked job C01 on swap, oversubscribe memory. However, the scheduler had no knowledge that it had swapped a parallel job and C01 remained on swap until the singletasked large job completed.

Autotasking reduced the elapsed time from 1463 seconds to 1231 seconds, a reduction of 16%.

The following figure compares the elapsed time of individual jobs in the 6.0 system. The figure indicates the elapsed time for each code for execution in the singletasked and autotasked workloads. Asterisks denote the autotasked jobs.

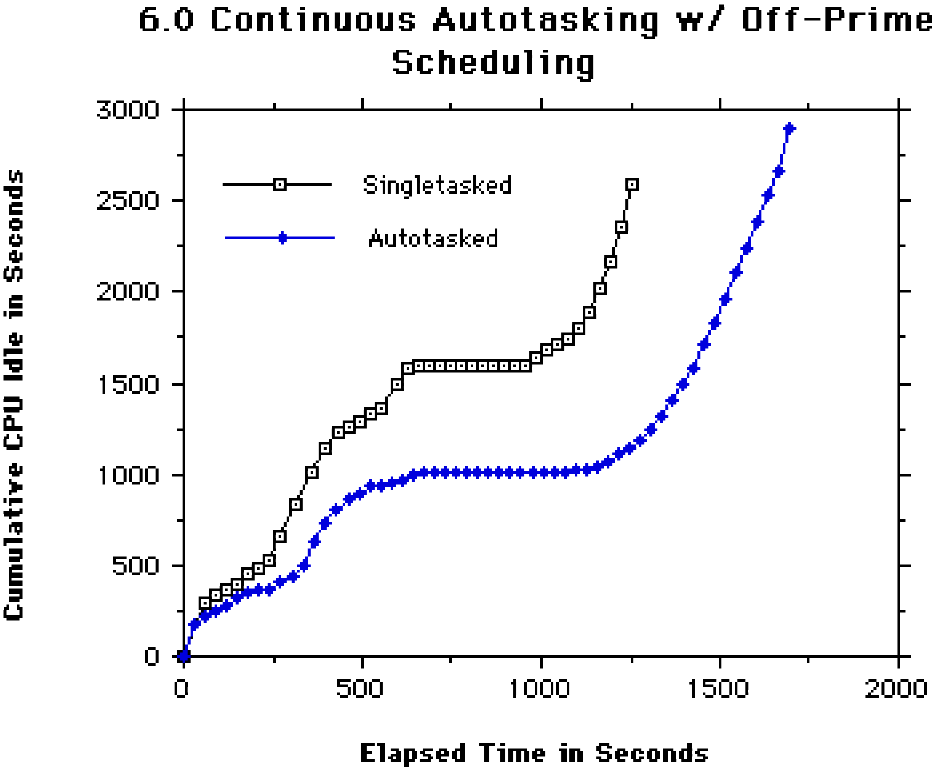


The figure indicates that the small memory autotasked jobs, C04, C05, C06, and C011, experience improved turnaround under the prime scheduler parameters. This improvement is obtained at the expense of five of the singletasked jobs.

3.3 UNICOS 6.0 Continuous Autotasking

Typical supercomputer workloads keep their processors continuously busy as opposed to the 1200-second model workload treated in the preceding sections. A supercomputer installation might average a single autotasked job in its workload, either by design or by user choice. This section addresses the performance gains observed by autotasking a single job continuously in the 12 job workload. In this case, the autotasked job is the efficient 8 MW C06 code described in the workload modelling section and the Off-Prime parameters provide the workload scheduling. To continuously resubmit the C06 code, the C06 script was modified to invoke a loop containing the C06 executable. This script continued to execute until all other programs had completed.

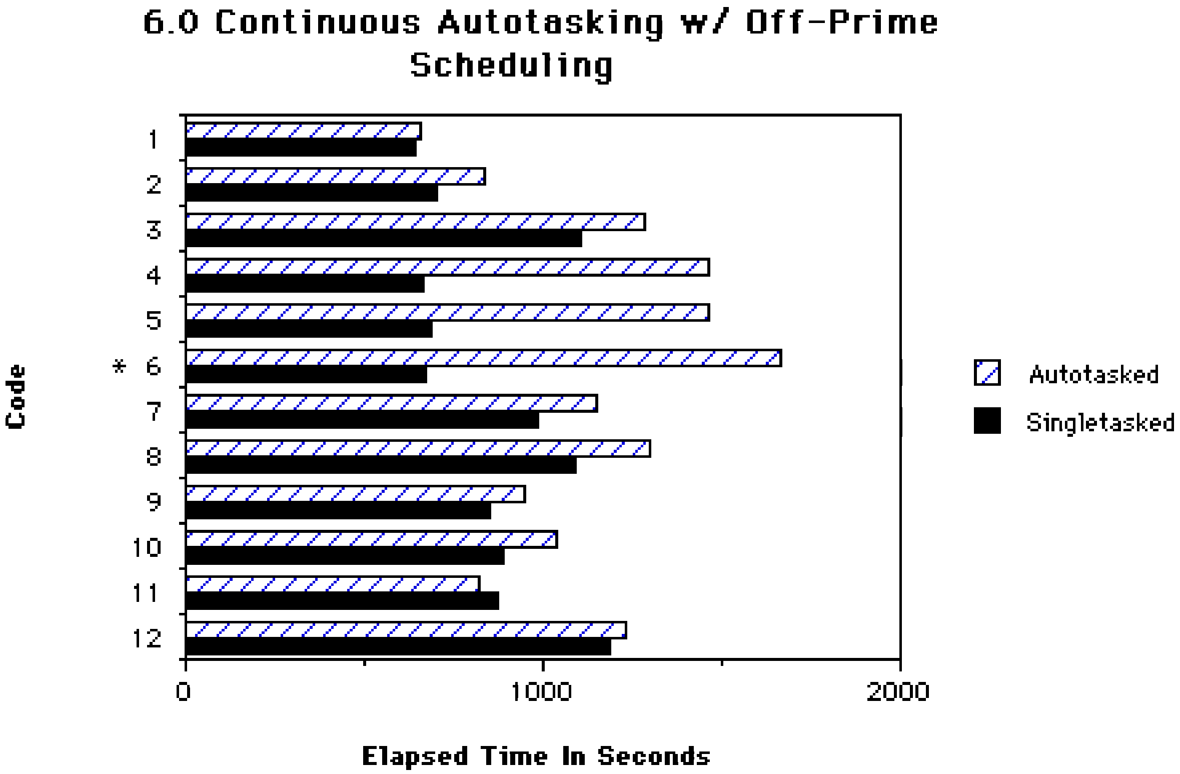
Autotasking large memory jobs might reduce idle more effectively, but NAS autotasking programs tend to average around 10 MW. NAS users with large memory requirements tend to store their arrays on SSD to increase their turnaround (Bergeron, 1990).



The above figure shows a three stage idle growth for the autotasked workload with an initial growth interval dominated by a singletasked job in memory, a period of zero growth corresponding to the completion of one big job, and an increase beyond 1300 seconds when the 59 MW autotasked job dominates execution. The rate of idle growth in the initial stage is about 1.3 idle seconds per elapsed second; Section 3.1.2 indicated the singletasked idle growth rate for this workload to be about 2.5 idle seconds per elapsed second. CPU memory starvation was responsible for the idle growth in the singletasked workload; autotasking a single job reduced the idle growth rate by a factor of 2. The high idle occurring at the end of elapsed time is an artifact of the 6.0 tendency for the slave processes to relinquish their CPUs. Autotasking a single job in an actual (many job) workload environment should prove more effective in limiting idle growth.

The total number of FLOPS for this workload was 1545 billion and workload elapsed time was 1664 seconds. The system performance rate for this case is 0.928 GFLOPS, an increase of 3% over singletasked throughput.

The next figure compares each job’s elapsed time in the singletasked workload with those in the continuous autotasking workload.



All but one job required more elapsed time in the autotasked workload, with elapsed time increases ranging from 2 to 120%. The very large increases in elapsed time were due to the Off-Prime tendency to keep jobs executing until completion.

This example employed an efficient job to utilize the idle, but not all autotasked jobs are efficient; in fact, workload throughput could decrease if the autotasked jobs represent low performance code (Bergeron, 1990).

3.4 UNICOS 6.0 System Effects

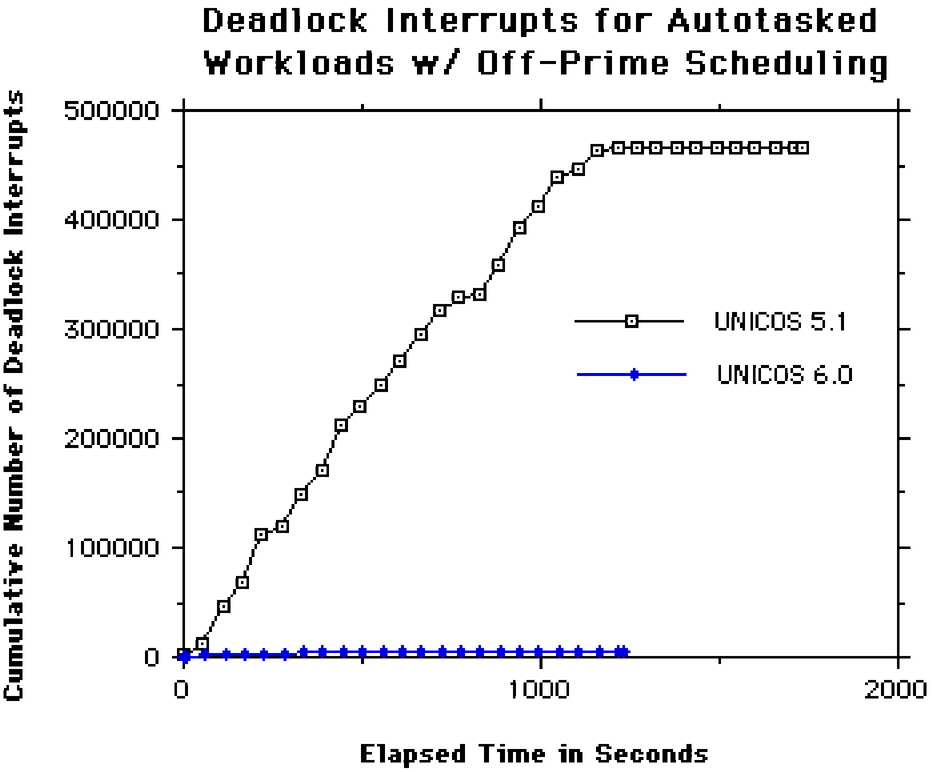
All autotasked workloads experience significant increases in system time due to the increased managerial burden placed upon the kernel in servicing the autotasked jobs. This section describes how UNICOS 6.0 improvements have reduced two aspects of this burden, deadlock interrupts and context switches.

Section 1.2 indicated that deadlock interrupts occur when the operating system must intercede in the normal process scheduling procedure because all autotasked processes associated with a given job wait on a semaphore. In a workload context, deadlock interrupt occurs because the kernel has preempted a CPU which was part of a multitasking group.

The 5.1 version of UNICOS allowed slave processes to keep their CPUs whenever the master task performed singletasked work. During such periods, these idle CPUs executed the semaphore wait test until the master task called for additional CPUs. Thus, CPUs were busy during a 5.1 autotasked job and the probability of kernel preemption of a CPU was high. Such preemptions would lead to a large number of deadlocks.

The 6.0 version of UNICOS required fewer interrupts because idle slave processes returned their CPUs to the operating system and went to sleep.

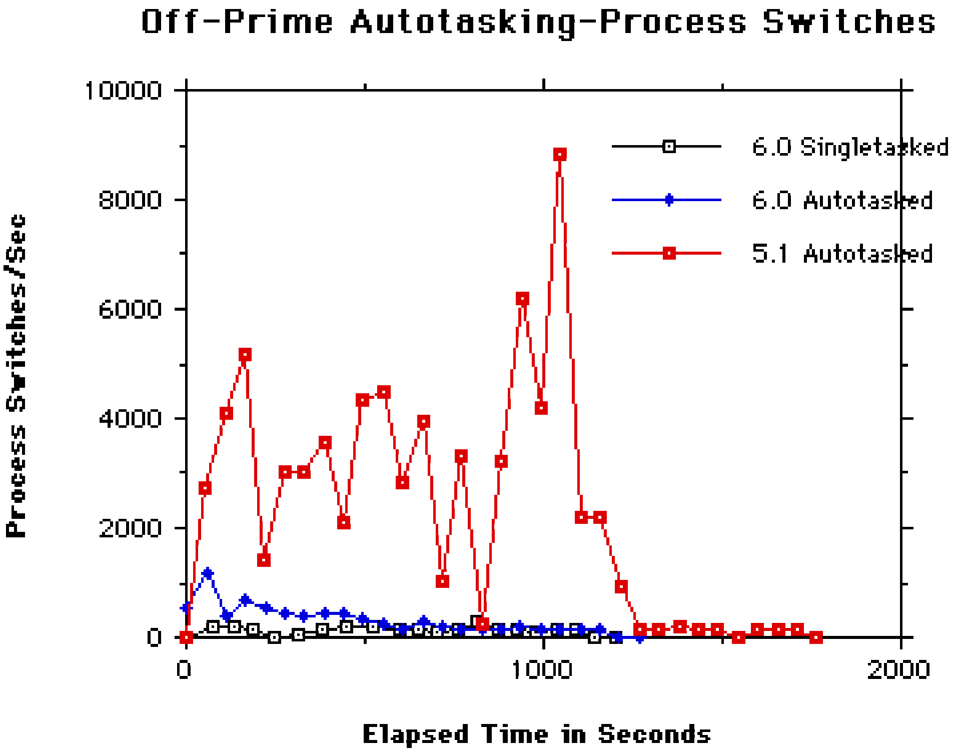
The following figure compares the cumulative number of deadlock interrupts for the two UNICOS versions executing the sample workload with the Off-Prime scheduling parameters.



The figure shows a reduction of almost 500,000 in the number of deadlock interrupts for the two workloads. Time spent processing deadlock interrupts is not recorded directly in the system call history, so it difficult to know whether this time exceeds the semaphore wait time.

One system effect which is recorded in the system traces is the context switch rate. UNICOS requires CPU time (charged as the increased system time) to transfer execution from one process to another. The process switch rate provides the exchange sequence rate, i.e., how often the operating system saves a current process, disconnects it from the CPU, arranges space for the next process and connects the new process to the CPU. Workload joblogs indicate that system time peaks during intervals of high context switching.

The following figure compares the process switch rate for three instances of workload execution with Off-Prime scheduler parameters: the 6.0 singletasked workload, the 6.0 autotasked workload, and the 5.1 autotasked workload.



The autotasked workloads display switch rates which are larger than those of the singletasked workload. UNICOS 5.1 switch rates are much higher when autotasked processes dominate execution The high 5.1 rates are another manifestation of the deadlock interrupts as frequent reschedulings require a large number of context switches. In the 6.0 autotasked workload, the autotasked processes reschedule themselves by invoking the system call *resch* . This approach reduces kernel involvement in the autotasked workload.

4.0 Discussion

These experiments indicate that autotasking a portion of a heterogeneous workload under UNICOS 6.0 can lead to a significant performance improvement.

The workload represented the NAS Y-MP batch workload with a small interactive content, a small oversubscription of memory, 90 MFLOPS per CPU, and scheduler parameters designed to promote throughput of computationally intense jobs. Since the NAS uses the first 30 MW of SSD as a cache for the swap disk, swapping of executables took place primarily between the SSD and main memory. The codes comprising the autotasking portion displayed single CPU performances exceeding 130 MFLOPS and dedicated speedups exceeding 3.

Table 3 shows workload elapsed time and its components: CPU, system and idle for the 6.0 and 5.1 versions of the operating system. The CPU times quoted in the table include semaphore wait time. The 6.0 1173 second elapsed time is 13% less than the 4-CPU 5.1 1352 second elapsed time and 30% less than the 8-CPU 1727 second elapsed time. Moreover, the data show that 6.0 autotasking provided a 4% increase in workload throughput whereas 5.1 autotasking led to throughput decreases.

Table 3. UNICOS Off-Prime Autotasking Summaries

UNICOS Version	6.0		5.1		
Scheduler	Off-Prime				
Workload	Single	Auto	Single	Auto	
NCPUS	N/A	8	N/A	8	4
Elapsed Time	1219	1173	1241	1727	1352
CPU Time	7319	7891	7309	10940	8003
System Time	96	177	95	1440	667
Idle Time	2337	1316	2572	1436	2146
System GFLOPS	0.903	0.939	0.887	0.637	0.814

The CPU times for the singletasked workloads correspond to the total CPU time required to execute the effectively twelve 600 second jobs comprising the workload. Autotasking produced noticeable increases in the CPU times due to the execution of the semaphore wait test by the slave processes comprising the autotasked jobs and the time spent executing extra autotasker-generated FORTRAN code. The above table indicates a 6% increase for UNICOS 6.0 and a 9% increase for the 4-CPU UNICOS 5.1 executions. The 8-CPU UNICOS 5.1 execution displays a 50% increase in CPU time.

System times for the singletasked workloads were about 7% of elapsed time, a value similar to those previously measured for the NAS workloads (Bergeron, 1990). Both versions of UNICOS produced large increases in autotasking system time with the 5.1 system time increasing by a factor of 7 to 14 and the 6.0 system time increasing by a factor of 2. A very large number of deadlock interrupts, caused by the use of a hardware semaphore to manage the parallel jobs, produced the 5.1 increases. Previous analyses indicated that process management by hardware semaphore gave unpredictable speedups when applied to autotasking single jobs in a workload (Carter, 1990). Comparison of system activity for the 6.0 workloads with that of the 5.1 workload showed that the autotasked workloads displayed large increases in process switch rates relative to their singletasked counterparts. This increase is in line with the larger number of processes present during the autotasked workloads.

UNICOS 5.1 autotasking was able to reduce the system idle for both 4-CPU and 8-CPU cases. Reduction of idle time is an insufficient criterion for autotasking efficiency since neither of these two cases could improve workload throughput. The 5.1 implementation kept the autotasked slave processes executing a semaphore wait test instead of sending them back to the operating system to perform useful work. The 6.0 version converted a large fraction of the idle to useful work and rescheduled unneeded CPUs back to the system.

Table 4 shows elapsed time and idle for 6.0 cases involving High Swap and Prime NAS scheduler parameters. Time constraints prevented execution of these cases under the 5.1 version of UNICOS.

Table 4. UNICOS Parametric Autotasking Summaries

UNICOS Version	6.0			
Scheduler	High Swap		Prime	
Workload	Single	Auto	Single	Auto
NCPUS	N/A	8	N/A	8
Elapsed Time	1600	1296	1463	1231
CPU Time	7266	8036	7272	7881
System Time	107	207	57	292
Idle Time	5427	2125	4375	1675
System GFLOPS	0.688	0.850	0.753	0.894

Adjustments in the 6.0 scheduler for high swap or interactive conditions increased the elapsed time improvements provided by autotasking. For the high swap case, autotasking produced a 19% improvement in elapsed time relative to its singletasked counterpart.

For the interactive scheduler parameters, autotasking led to a 16% improvement in performance. The autotasked execution showed that UNICOS may leave autotasked jobs on the swap disk when it might be profitable to bring them into execution. The UNICOS scheduler has no knowledge that a given job is parallel as it places the job on the run queue for execution.

Table 5 shows that autotasking a single job continuously in the model workload led to a 3% improvement in system GFLOPS. The 8 MW autotasked job performed at a single CPU rate about double the workload CPU rate and displayed a parallel processing efficiency of 44% during dedicated runs. This performance might approximate that of a typical production code. Autotasking purchased this performance increase at the expense of a 15% increase in the elapsed time of the remaining singletasked jobs in the workload.

Autotasking actually provided a somewhat larger performance increase than the above method indicates as can be shown by considering only the time during which other jobs are also executing. The reason for this additional consideration arises because C03 executing alone on 8 CPUs performs at 0.652 GFLOPS and this rate is about 30% below the workload average. Excluding the time during which C03 performed alone in the autotasked workload gives a 7% performance improvement due to continuous autotasking.

Table 5. UNICOS Continuous Autotasking Summaries

UNICOS Version	6.0	
Scheduler	Off-Prime	
Workload	Single	Auto
NCPUS	N/A	8
Elapsed Time	1219	1664
CPU Time	7319	10235
System Time	96	200
Idle Time	2337	2877
Workld GFLOPS	1101	1545
System GFLOPS	0.903	0.928

These autotasked performance increases (4-19%) are somewhat smaller than those reported for microtasking on an homogeneous X-MP workload (Bieterman, 1987). The X-MP workload heavily oversubscribed both memory and CPU by a factor of 3; moreover, the X-MP system swapped directly to a DD-49. These factors could be the source of a much larger amount of idle time than was produced in the current analysis; such larger time provides the potential for large performance improvements. Finally, microtasking provides a command for releasing idle CPUs and this command could allow a more efficient use of idle processors.

5.0 Conclusions

Integration of autotasked jobs into the NAS workload must occur. While performance considerations imply that succeeding generations of supercomputers will contain an increasing number of processors, cost considerations limit the rate of high speed memory growth. Jobs requiring large amounts of high speed memory must make use of additional processors to limit memory residence time.

A series of model workload executions has indicated that the 6.0 version of UNICOS allowed autotasking to improve performance. Since these workloads performed at 90 MFLOPS/CPU prior to autotasking, this increase in performance is significant. Workload performance improved because autotasked jobs were able to transform idle cycles into cycles performing useful work.

The magnitude of the autotasking performance increase depended upon the amount of idle time, the characteristics of the autotasked jobs, and the scheduler parameters. The workload experiments contained herein displayed throughput improvements of 4 to 19%.

The previous 5.1 version of UNICOS allowed no performance improvement under autotasking for a variety of scheduler parameters.

The workload mix could be improved to represent the NAS workload more faithfully. The improvements would include a greater emphasis on SSD utilization and a better simulation of system idle. Adjusting the mix of jobs would resolve the first difficulty. Imposing a relatively constant idle on the workload requires a background mix of jobs which would execute before and after the autotasked jobs executed. This approach would provide a better approximation to the NAS production environment.

Examination of the model workload runlogs revealed that autotasked jobs remained on the swap disk during periods when CPUs idled for lack of memory. A modification to the UNICOS scheduler which boosted the priority of autotasked jobs on the run queue during prolonged periods of idle could provide performance improvement. Other improvements including Gang scheduling of parallel processes are possible (Seager and Stichnoth, 1989).

The scheduler improvements described above assume a supply of parallel jobs. Although experiments indicate that an autotasked job in the NAS 6.0 UNICOS workload enjoys a 50% reduction in elapsed time (Carter 1991), this reduction has produced a negligible amount of user parallel processing. A stronger motivation is required to coax user autotasking.

A stronger incentive for user autotasking would arise through the dedication of a cluster of CPUs to parallel processing. A parallel job would have a guaranteed access to the group and the kernel could not preempt the CPUs in this cluster. CPUs in the cluster would run singletasked jobs in the absence of parallel jobs. Administrators could adjust the number of CPUs in the parallel cluster. Elapsed time reduction factors of 2 and larger could eliminate the overnight wait for a production job.

6.0 References

R. Bergeron (1990) “Performance Analysis of the NAS Y-MP Workload,”NAS RND Technical Report RND 90-009.

M. Bieterman (1987), “The Impact of Microtasked Applications In A Multiprogramming Environment,” Proceedings of the Cray User Group, October, 1987, Bologna.

R. Carter (1990), “Autotasked Performance of a NASA CFD Code,” NAS RND Technical Report RND 90-003.

R. Carter (1991), “Autotasked Performance of a NASA CFD Code in the NASA NAS Production Environment,” NAS RND Technical Report (forthcoming).

Cray Research Inc., *Cray Y-MP and Cray X-MP Multitasking Programmer's Manual*, Pub. No. SR-022E, Cray Research Inc.,1988.

R. Ciotti (1990), NAS RND, private communication.

S. Reinhardt (1985), “A Data-Flow Approach to Multitasking on CRAY X-MP Computers,” Proceedings of the Tenth ACM Symposium on Operating System Principles, December 1985, pp 107-114.

M. Seager and J. Stichnoth, “Simulating the Scheduling of Parallel Supercomputer Applications,” UCRL-102059, Lawrence Livermore Laboratory, Livermore, Ca., 1989.

Appendix A

The Off-Prime and High Swap Scheduler Parameters represent variations on the Prime Parameters. NAS administrators derived the Prime Parameters by examining the swap queue priorities obtained from the system core image with UNICOS tools such as crash. NAS administrators continue to tune these values. The parameters do not distinguish autotasked processes from singletasked processes.

Prime Scheduling Parameters.

[-H] hog_max_mem...107.4 MW

[-h] memhog.................4.9 MW

[-c] cpuhog....................90 Secs

[-f] fit_boost..............-2.

[-M] mfactor_in.........3200.[-m] mfactor_out.....-3200.

[-T] tfactor_in...........-1.[-t] tfactor_out........1.

[-P] pfactor_in...........3.[-p] pfactor_out.......3.

[-N] nfactor_in...........0.[-n] nfactor_out.......0.

[-G] in_guarantee......5.,60 [-g] out_guarantee..0.,10

[-K] constant_in........0.[-k] constant_out....0.

[-R] thrash-inter.......0.[-B] thrash-blks.......0

[-C] compress_intv..30.[-r] cpu_factor.........24

[-L] big proc................32000[-Z] itime....................5

[-z] smallproc............2000 [-x] max_outage.......0

[-i] intrctve prfrd.....1

Off-Prime Scheduling Parameters

The major change to the Prime Parameters is the removal of restrictions on the memory and cpu. Setting H, h, and c to 0 achieves this. Also interactive processes no longer enjoy preferred status.

```
[-H] hog_max_mem...0.0 MW
[-h] memhog.................0.0 MW
[-c] cpuhog....................0.0 Secs
[-f] fit_boost..............-2.
[-M] mfactor_in.........3200.[-m] mfactor_out.....-3200.
[-T] tfactor_in...........-1.[-t] tfactor_out........1.
[-P] pfactor_in...........1.[-p] pfactor_out.......1.
[-N] nfactor_in...........0.[-n] nfactor_out.......0.
[-G] in_guarantee......0.,60 [-g] out_guarantee..0.,0.
[-K] constant_in........0.[-k] constant_out....0.
[-R] thrash-inter.......0.[-B] thrash-blks.......0
[-C] compress_intv..30.[-r] cpu_factor.........24
[-L] big proc.................32000[-Z] itime....................5
[-z] smallproc.............2000 [-x] max_outage.......0
[-i] intrctve prfrd.....1
```

High Swap Scheduling Parameters

The High Swap parameters are obtained by removing the big proc restriction, i.e. setting L equal to 0. This action ensured that all jobs obtained an equivalent amount of CPU time for the 12-job workload.

```
[-H] hog_max_mem...0.0 MW
[-h] memhog................0.0 MW
[-c] cpuhog....................0.0 Secs
[-f] fit_boost.............-2.
[-M] mfactor_in.........0.0[-m] mfactor_out.....0.0
[-T] tfactor_in...........-1.[-t] tfactor_out........1.
[-P] pfactor_in...........1.[-p] pfactor_out.......1.
[-N] nfactor_in...........0.[-n] nfactor_out.......0.
[-G] in_guarantee......0.,60 [-g] out_guarantee..0.,0.
[-K] constant_in........0.[-k] constant_out....0.
[-R] thrash-inter.......0.[-B] thrash-blks.......0
[-C] compress_intv..0.0[-r] cpu_factor.........24
[-L] big proc................0[-Z] itime....................5
[-z] smallproc............2000 [-x] max_outage.......0
[-i] intrctve prfrd.....0
```

..